This project contains 2 main classes - server and proxy and 4 other helper classes – CacheHandler.java, FileMetadata.java, FileDetails.java and ServerService.java.

FileDetails is a serializable object, used to send file details (valid flag, length and version) from server to client.

CacheHandler implements cache operations.

FileMetadata object contains file details of the files on the cache. Object for each file is stored in the FileVersionTable Hashmap, the key for each object being the file name.

ServerService.java is the RMI interface used by the server that enables the proxy to make remote calls to the server.

I have a hashmap <String, FileMetadata> that stores the file details of each file on cache.

FileMetadata has the following members –

```
int fileVersion – version of the file
String filename – name of the original file of the file copy
int clientCount – number of clients so far (used to name writer's copy)
int writerCount – number of writers
int readerCount – number of readers
long lastAccessTime – Latest time of access
String nextVersionName – name of the newer version of the file (null if no
later version)
```

The FD table for each client is a hashmap with key as FD and value as FileObject. The fileObject has the following members –

```
RandomAccessFile file
int option – READ/WRITER/ISDIRECTORY permission
String path – Name of the file
```

When a client call open on a file it makes the following checks-

Step 1. Call RMI checkFile() fuction to server – server returns the following values -

Length – length of the file if file exists
-1 – if file does not exist on server
-2 – if path is a directory
Version - version of file
0 if file does not exist / or is directory

Step 2. If file does not exist on server – proxy throws error

Else –
Check if file is present of cache and is the latest version

Step 3. If the file does not exist on cache – download it, update the FileVersionTable and update freespace value in cache.

If the file on cache is fresh, use it as readers copy or clone the copy for a writer.

If the files is an older version, check if there is a newer version on the cache. This can be checked by the checking the field `nextVersionName.` If this field is not populated, there is no later copy of the file on the cache. If the newer version exists we do not download the latest version from the file.

The writer's copy of the file keeps track of it's original copy by the field `filename.`

Everytime a file is accessed the `lastAccessTime` is updated, which will be used in LRU eviction.

Cloned file names – original file name is appended with the noOfClients member of the original fileMetadata. This number is always unique and is increasing with every reader or writer accessing the object.

When the file is closed :

Remove the fd entry from the FD table, update the reader and writer count in the FileMetadata object and take the following action:

1. Writers copy –
    i. Replace the original file on the server with the latest one
    ii. If there is no reader on original file on cache, replace it on cache and delete the writer's copy
    iii. If there is a reader on original copy, mark the writer's copy as the latest version on cache
2. Reader copy –
    i. Check if the reader was the last one and there exists a newer version on the cache, replace the newer version as the original version on cache.
    ii. Close the file

The Cache object maintains the FileVersionTable and the freeSpace available in cache.

Every time before a file is downloaded or cloned, I check the length of the file with the freeSpace. If the space is lesser, I call the evict() function. This linearly searches the FileVersionTable for the least recently used file, which does not have any reader or writers. This search is continued until the available free space is more equal to or more than the file to be downloaded.

If the evict() is unable to make enough space for the file it return -1, which in turn causes the proxy to return ENOMEM error code.

I have used intrinsic lock object – cachelock and synchronized statement to synchronize access to cache objects.

The clientdone() method clears the fdTable for each client, closes it's open files and updates the FileVersionTable entries accordingly.

The server maintains a HashMap – FileVersionTable that stores the version on each file on the server.